

# JSIQA

*(Jessica)*

# JavaScript IQ Processing AARONIA

## General

This document matches the Aaronia RTSA Suite Pro Built number 11738 and above.

## Additional Datatypes

JSIQA extends the common JavaScript numeric type into the complex realm. Numbers can be real, imaginary or a combination. An imaginary number may be the result of a numeric operation, such as the square root of minus one, or a literal with the ending character “i”, such as “123i” or “3.4e5i”.

## System Objects

System objects are provided by the runtime system, and need not be imported before use.

## IQ Array Processing – IQ

The IQ library/object provides optimized routines for manipulating typed and untyped arrays of real and complex numbers in a vector style.

```
const a = [1, 2, 3, 4, 5];  
let b = IQ.add(a, 4);
```

The returned array is usually similar to the provided array typed or untyped.

### IQ.abs(data)

<b>data</b>	Array, TypedArray, Number	Source data
-------------	---------------------------	-------------

Returns an array with the absolute values of the source data array.

### IQ.phi(data)

<b>data</b>	Array, TypedArray, Number	Source data
-------------	---------------------------	-------------

Returns an array with the angle to the real value axis for the complex values of the source data array.

### IQ.exp(data)

<b>Data</b>	Array, TypedArray, Number	Source data
-------------	---------------------------	-------------

Returns an array with the natural exponentiation of the source array.

### IQ.log(data)

<b>data</b>	Array, TypedArray, Number	Source data
-------------	---------------------------	-------------

Returns an array with the natural logarithm of the source array.

### IQ.i(data)

<b>data</b>	Array, TypedArray, Number	Source data
-------------	---------------------------	-------------

Returns the in phase (real) component of the complex values

**IQ.q(data)**

<b>data</b>	Array, TypedArray, Number	Source data
-------------	---------------------------	-------------

Returns the out of phase (imaginary) component of the complex values.

**IQ.dphi(data)**

<b>data</b>	Array, TypedArray	Source data
-------------	-------------------	-------------

Returns an array with the change of angle to the real value axis for the complex values of the source data array.

**IQ.fft(data)**

<b>data</b>	Array, TypedArray	Source data
-------------	-------------------	-------------

Forward Fourier-transformation of a complex array.

**IQ.ifft(data)**

<b>data</b>	Array, TypedArray	Source data
-------------	-------------------	-------------

Inverse Fourier-transformation of a complex array.

**IQ.sum(data)**

<b>data</b>	Array, TypedArray	Source data
-------------	-------------------	-------------

Returns the sum of all elements in the array

**IQ.min(data1, data2)**

<b>data1</b>	Array, TypedArray, Number	Source data
<b>data2</b>	Array, TypedArray, Number	Source data

Returns an array with the smaller elements of the two source arrays

**IQ.max(data1, data2)**

<b>data1</b>	Array, TypedArray, Number	Source data
<b>data2</b>	Array, TypedArray, Number	Source data

Returns an array with the larger elements of the two source arrays

**IQ.conj(data)**

<b>data</b>	Array, TypedArray, Number	Source data
-------------	---------------------------	-------------

Returns the complex conjugated values in an array or typed array

**IQ.add(data1, data2)**

<b>data1</b>	Array, TypedArray, Number	Source data
<b>data2</b>	Array, TypedArray, Number	Source data

Returns the elementwise sum of the two source arrays

**IQ.sub(data1, data2)**

<b>data1</b>	Array, TypedArray, Number	Source data
<b>data2</b>	Array, TypedArray, Number	Source data

Returns the element wise difference of the two source arrays

**IQ.mul(data1, data2)**

<b>data1</b>	Array, TypedArray, Number	Source data
<b>data2</b>	Array, TypedArray, Number	Source data

Returns the element wise product of the two source arrays

**IQ.div(data1, data2)**

<b>data1</b>	Array, TypedArray, Number	Source data
<b>data2</b>	Array, TypedArray, Number	Source data

Returns the element wise quotient of the two source arrays

**IQ.and**(data1, data2)

<b>data1</b>	Array, TypedArray, Number	Source data
<b>data2</b>	Array, TypedArray, Number	Source data

Returns the element wise binary and of the two source arrays

**IQ.or**(data1, data2)

<b>data1</b>	Array, TypedArray, Number	Source data
<b>data2</b>	Array, TypedArray, Number	Source data

Returns the element wise binary or of the two source arrays

**IQ.xor**(data1, data2)

<b>data1</b>	Array, TypedArray, Number	Source data
<b>data2</b>	Array, TypedArray, Number	Source data

Returns the element wise binary exclusive or of the two source arrays

**IQ.andnot**(data1, data2)

<b>data1</b>	Array, TypedArray, Number	Source data
<b>data2</b>	Array, TypedArray, Number	Source data

Returns the element wise binary and of the first and the inverted second array

**IQ.eq**(data1, data2)

<b>data1</b>	Array, TypedArray, Number	Source data
<b>data2</b>	Array, TypedArray, Number	Source data

Returns the element wise comparison for equal of the two source arrays

**IQ.ne**(data1, data2)

<b>data1</b>	Array, TypedArray, Number	Source data
<b>data2</b>	Array, TypedArray, Number	Source data

Returns the element wise comparison for not equal of the two source arrays

**IQ.lt**(data1, data2)

<b>data1</b>	Array, TypedArray, Number	Source data
<b>data2</b>	Array, TypedArray, Number	Source data

Returns the element wise comparison for less than of the two source arrays

**IQ.gt**(data1, data2)

<b>data1</b>	Array, TypedArray, Number	Source data
<b>data2</b>	Array, TypedArray, Number	Source data

Returns the element wise comparison for greater than of the two source arrays

**IQ.le**(data1, data2)

<b>data1</b>	Array, TypedArray, Number	Source data
<b>data2</b>	Array, TypedArray, Number	Source data

Returns the element wise comparison for less than or equal of the two source arrays

**IQ.ge**(data1, data2)

<b>data1</b>	Array, TypedArray, Number	Source data
<b>data2</b>	Array, TypedArray, Number	Source data

Returns the element wise comparison for greater than or equal of the two source arrays

**IQ.if**(cond, if, else)

<b>cond</b>	Array, TypedArray, Number	Condition
<b>if</b>	Array, TypedArray, Number	Values for true condition
<b>else</b>	Array, TypedArray, Number	Values for false condition

Returns an array with the elements of either the “if” or the “else” source argument, based on the contents of the “cond” argument.

**IQ.gather(data, index)**

<b>data</b>	Array, TypedArray	Source data
<b>index</b>	Array, TypedArray	Index data

Returns an array with the size of the “index” argument and the values of the “data” argument indexed by the values of the “index” argument.

**IQ.modulate(data, phi, dphi)**

<b>data</b>	Array, TypedArray	Source data
<b>phi</b>	Complex	Phase offset
<b>dphi</b>	Complex	Phase shift

Returns an array with the source values multiplied by phi and the powers of dphi.

**IQ.resample(data, offset, step)**

<b>data</b>	Array, TypedArray	Source data
<b>offset</b>	Number	Start offset
<b>step</b>	Number	Offset step

Returns an array with resampled values at the starting offset and offset plus integer multiples of the sept value. Offset and step may be non integer values.

**IQ.rankIndex(data, count, down)**

<b>data</b>	Array, TypedArray	Source data
<b>count</b>	Integer	Number of elements to return
<b>down</b>	Boolean	Sort from highest to lowest

Returns an array with the indices of the sorted smallest (or highest) elements on a float array.

**IQ.nth(data, n);**

<b>data</b>	Array, TypedArray	Source data
<b>n</b>	Number	Element index

Returns the n-th element from the sorted source data array.

**IQ.zcross**(data, [thresh])

<b>data</b>	Array, TypedArray	Source data
<b>thresh</b>	Number	Optional threshold value

Returns an array of the indices of all elements in the source data array, that cross the threshold value.

**IQ.zfcross**(data, [thresh])

<b>data</b>	Array, TypedArray	Source data
<b>thresh</b>	Number	Optional threshold value

Returns an array of the of the linear interpolated positions where the values of the data array cross the threshold value.

## XML Parser – XML

Parsing and manipulating XML documents.

**XML.parse**(text)

<b>text</b>	String	XML Format
-------------	--------	------------

Returns an xml document as a tree of XML objects.

**XML.prototype.node**(tag)

<b>tag</b>	String	Name of an XML element
------------	--------	------------------------

Returns the first child element with the given tag.

**XML.prototype.nodes**(tag)

<b>tag</b>	String	Name of an XML element
------------	--------	------------------------

Returns an array of child elements with the given tag.



**XML.prototype.text()**

--	--	--

Returns the text of the XML object

**XML.tag**

Tag of the XML element

**XML.attributes**

Object with the XML element attributes

**XML.elements**

Array of the XML child elements.

[System / Application log – syslog](#)

Writing into the application log.

**syslog.debug(value, ...)**

<b>value</b>	Value	Values to log
--------------	-------	---------------

Send a debug message to the log file

**syslog.warning(value, ...)**

<b>value</b>	Value	Values to log
--------------	-------	---------------

Send a warning message to the log file

**syslog.info(value, ...)**

<b>value</b>	Value	Values to log
--------------	-------	---------------

Send an info message to the log file

**syslog.critical(value, ...)**

<b>value</b>	Value	Values to log
--------------	-------	---------------

Send a critical message to the log file

## UTF8 Encoding – UTF8

Formatting and parsing of UTF8 data to and from string objects.

### UTF8.encode(string, [zero])

<b>string</b>	String	Source string
<b>zero</b>	Boolean	Append a zero byte to the characters in the buffer

Encodes a string into an array buffer using UTF8 encoding

### UTF8.decode(data)

<b>data</b>	ArrayBuffer	UTF8 encoded string
-------------	-------------	---------------------

Decodes the UTF8 encoded string in the given array buffer.

## Libraries

### Promises and Callbacks – “promise.js”

Promises are structured way to avoid callbacks. A function that would normally accept a callback returns a promise object if no callback is provided. The promise object can be used to continue processing when the asynchronous component of the function completes.

Most library functions are asynchronous, they either accept a callback function that is called with the result of the function or return a promise, if no callback function is provided.

### Promise(resolver)

<b>resolver</b>	function(resolve, reject)	Resolver function
-----------------	---------------------------	-------------------

Creates a new Promise object with a given resolver function. The resolver function is called with two functions, a resolve and a reject function. The resolver must call one of those two functions when the asynchronous call completes.

**Promise.callback(resolve, reject)**

<b>resolve</b>	function(value)	Function to be called if successful
<b>reject</b>	function(err)	Function to be called if failed

Creates a node.js style callback function from the given promise callbacks. This function can be used to provide a promise compatible implementation for a function that normally uses node.js callback style completion.

**Promise.resolve(value)**

<b>value</b>		Result
--------------	--	--------

Create an already resolved promise with the given value as return value.

**Promise.reject(err)**

<b>err</b>	Error	Failure reason
------------	-------	----------------

Create an already rejected promise with the given error reason

**Promise.all(coll)**

<b>coll</b>	Array or Object	Collection of promises
-------------	-----------------	------------------------

Creates a promise for a collection of promises that is resolved when all promises of the collection are resolved or rejected when at least one of the promises is rejected.

**Promise.prototype.done(value)**

<b>value</b>		Result
--------------	--	--------

Resolve the given promise, like calling the resolve function in the resolver.

### Promise.prototype.fail(err)

<b>err</b>	Error	Failure reason
------------	-------	----------------

Reject the given promise, like calling the reject function in the resolver.

### Promise.prototype.then(cbdone, [cbfail])

<b>cbdone</b>	function(value)	Function to be called when the promise is resolved
<b>cbfail</b>	function(err)	Function to be called when the promise is rejected

This method is the final client part of a promise. The callback functions will be executed immediately if the promise is already resolved or rejected. Otherwise they will be called when the promise is resolved or rejected. The function returns a promise itself, which can be used to chain promises. The returned promise is considered resolved, when the callback functions return with a normal value. The callback functions may also provide a promise as a return value, which will then decide the fate of the initially returned promise.

### Promise.prototype.catch(cbfail)

<b>cbfail</b>	function(err)	Function to be called when the promise is rejected
---------------	---------------	--

This method is like the “then” method but provides only a fail callback.

### Promise.prototype.finally(cb)

<b>cbfail</b>	function()	Function to be called when the promise is rejected or resolved
---------------	------------	--

This method is like the “then” method but uses the same callback for success and failure.

## Asynchronous Processing – “async.js”

### Async.each(coll, iteratee, [cb])

<b>coll</b>	Array	Collection
-------------	-------	------------

<b>iteratee</b>	function(value, cb)	Iterator function called for each element
<b>cb</b>	function(err)	Optional callback

Call the iterator function for each element of the collection. The iterator must call the provided callback when finished. The optional callback will be called, when all iterations are completed. The callback will receive the error message of the first iteration that returned an error in its callback.

**Async.eachOf(coll, iteratee, [cb])**

<b>coll</b>	Array	Collection
<b>iteratee</b>	function(value, index, cb)	Iterator function called for each element
<b>cb</b>	function(err)	Optional callback

Call the iterator function for each element of the collection. The iterator must call the provided callback when finished. The optional callback will be called, when all iterations are completed. The callback will receive the error message of the first iteration that returned an error in its callback.

**Async.every(coll, iteratee, [cb])**

<b>coll</b>	Array	Collection
<b>iteratee</b>	function(value, cb)	Iterator function called for each element
<b>cb</b>	function(err, value)	Optional callback

Call the iterator function for each element of the collection. The iterator must call the provided callback when finished, providing an additional Boolean value. The optional callback will be called, when all iterations are completed. The callback will receive the error message of the first iteration that returned an error in its callback. The final return value will be true, if all iterations returned true.

**Async.some(coll, iteratee, [cb])**

<b>coll</b>	Array	Collection
<b>iteratee</b>	function(value, cb)	Iterator function called for each element
<b>cb</b>	function(err, value)	Optional callback

Call the iterator function for each element of the collection. The iterator must call the provided callback when finished, providing an additional Boolean value. The optional callback will be called, when all iterations are completed. The callback will receive the error message of the first iteration that returned an error in its callback. The final return value will be true, if at least one iteration returned true.

**Async.filter(coll, iteratee, [cb])**

<b>coll</b>	Array	Collection
<b>iteratee</b>	function(value, cb)	Iterator function called for each element
<b>cb</b>	function(err, value)	Optional callback

Call the iterator function for each element of the collection. The iterator must call the provided callback when finished, providing an additional Boolean value. The optional callback will be called, when all iterations are completed. The callback will receive the error message of the first iteration that returned an error in its callback. The final return value will be an array with all elements of the original collection, for which the iteratee return true.

#### **Async.map(coll, iteratee, [cb])**

<b>coll</b>	Array	Collection
<b>iteratee</b>	function(value, cb)	Iterator function called for each element
<b>cb</b>	function(err, value)	Optional callback

Call the iterator function for each element of the collection. The iterator must call the provided callback when finished, providing a mapped version of the input value. The optional callback will be called, when all iterations are completed. The callback will receive the error message of the first iteration that returned an error in its callback. The final return value will be an array with all elements returned by the iterator functions.

#### **Async.mapValues(coll, iteratee, [cb])**

<b>coll</b>	Array	Collection
<b>iteratee</b>	function(value, index, cb)	Iterator function called for each element
<b>cb</b>	function(err, value)	Optional callback

Call the iterator function for each element of the collection. The iterator must call the provided callback when finished, providing a mapped version of the input value. The optional callback will be called, when all iterations are completed. The callback will receive the error message of the first iteration that returned an error in its callback. The final return value will be an array with all elements returned by the iterator functions.

#### **Async.parallel(tasks, [cb])**

<b>tasks</b>	Array of function(cb)	Collection of functions
<b>cb</b>	function(err, value)	Optional callback

Call all functions of the tasks collection in parallel. The final callback will be invoked, when all tasks have completed or at least one failed. The result is an array of all the values generated and returned by the tasks.

### **Async.whilst**(test, iteratee, [cb])

<b>test</b>	function()	Check for condition
<b>iteratee</b>	function(value, index, cb)	Iterator function called for each iteration
<b>cb</b>	function(err, value)	Optional callback

Calls the test function and the iteratee function as long as test returns true. The callback will be made with the final results of the iteratee callback.

### **Async.until**(test, iteratee, [cb])

<b>test</b>	function()	Check for condition
<b>iteratee</b>	function(value, index, cb)	Iterator function called for each iteration
<b>cb</b>	function(err, value)	Optional callback

Calls the iteratee function and the test function as long as test returns true. The callback will be made with the final results of the iteratee callback.

### **Async.forever**(iteratee, [cb])

<b>iteratee</b>	function(cb)	Iterator function called for each iteration
<b>cb</b>	function(err, value)	Optional callback

Calls the iterate function with a callback as argument. The iterate calls the callback when finished to start the next iteration. The iteration stops, if a non null argument is passed to the callback as an error.

## File System – “fs.js”

The file system library provides access to system files in an asynchronous way. Files operate on byte buffers or typed arrays. The result of the file operation can be received by either a callback or a promise.

### **File.read**(fname, [cb])

<b>fname</b>	String	Path to the file
<b>cb</b>	function(err, buffer)	Optional callback

Read the content of the file with the given name into an ArrayBuffer.

Example writing the content of a text file to the console using a promise as the result.

```
import { File } from "fs.js"  
File.read("e:/tmp/test.txt").then(d => {
```

```
    console.log(UTF8.decode(d));
  });
```

**File.write**(fname, buffer, [cb])

<b>fname</b>	String	Path to the file
<b>buffer</b>	ArrayBuffer / Array / TypedArray	Data to write
<b>cb</b>	function(err)	Optional callback

Writes the content of the buffer into the file. Creates a new file or truncates an existing file before writing.

```
import { File } from "fs.js"
let d = new Float32Array(1000);
for(let i=0; i<d.length; i++)
    d[i] = Math.cos(i);
File.write("e:/tmp/floats.bin", d);
```

**File.append**(fname, buffer, [cb])

<b>fname</b>	String	Path to the file
<b>buffer</b>	ArrayBuffer / Array / TypedArray	Data to write
<b>cb</b>	function(err)	Optional callback

Writes the content of the buffer into the file. Creates a new file or appends the data to an existing.

**File.exists**(fname, [cb])

<b>fname</b>	String	Path to the file
<b>cb</b>	function(err, bool)	Optional callback

Returns true if the given file (or folder) exists

**File.mkdir**(path, [cb])

<b>path</b>	String	Path to the directory
<b>cb</b>	function(err)	Optional callback

Creates a the complete path for the given directory name



**File.delete(path, [cb])**

<b>path</b>	String	Path to the file to delete
<b>cb</b>	function(err)	Optional callback

Delete a file

**File.rename(path, target, [cb])**

<b>Path</b>	String	Path to the file to rename
<b>Target</b>	String	Target path of the file to rename
<b>cb</b>	function(err)	Optional callback

Rename a file

**File.files(path, pattern, recursive, [cb])**

<b>path</b>	String	Path to the directory
<b>pattern</b>	String	Pattern to match
<b>recursive</b>	Bool	Search all child folders recursive
<b>cb</b>	function(err, files)	Optional callback

Returns an array of objects describing the matching files

**File.list(path, pattern, recursive, [cb])**

<b>path</b>	String	Path to the directory
<b>pattern</b>	String	Pattern to match
<b>recursive</b>	Bool	Search all child folders recursive
<b>cb</b>	function(err, files)	Optional callback

Returns an array of matching file paths

**File.open(fname, [cb])**

<b>fname</b>	String	Path to the file
<b>cb</b>	function(err, file)	Optional callback

Open an existing file for reading and create a file object.

**File.create**(fname, [cb])

<b>fname</b>	String	Path to the file
<b>cb</b>	function(err, file)	Optional callback

Create a new file or truncate an existing file for writing and create a file object.

**File.prototype.close**([cb])

<b>cb</b>	function(err)	Optional callback
-----------	---------------	-------------------

Close a file object and commit pending writes to storage.

**File.prototype.seek**(offset, [cb])

<b>offset</b>	Number	Offset in file
<b>cb</b>	function(err)	Optional callback

w

Move file read / write pointer to the given byte offset in the file.

**File.prototype.read**(size, [cb])

<b>size</b>	Number	Size in bytes
<b>cb</b>	function(err, buffer)	Optional callback

Read the number of bytes from the file into an ArrayBuffer. The resulting buffer may be smaller than the given size. The result will be an empty ArrayBuffer when reaching then end of the file.

Iterating over a file is best done using an async function.

```
import { File } from "fs.js"
async function sumFile(name) {
  let f = await File.open(name);
  let a, sum = 0;
  while ((a = await f.read(4)).byteLength == 4) {
    sum += new DataView(a).getFloat32(0, true);
  }
  return sum;
}
sumFile("e:/tmp/floats.bin").then(sum => {
  console.log("sum", sum);
}).catch(err => {
  console.log("Error", err);
});
```

```
});
```

**File.prototype.write**(buffer, [cb])

<b>buffer</b>	ArrayBuffer / Array / TypedArray	Data to write
<b>cb</b>	function(err)	Optional callback

Write the data from the buffer into the file.

TCP Client Sockets – “tcp.js”

**TCP.adapters**([cb])

<b>cb</b>	function(err, socket)	Optional callback
-----------	-----------------------	-------------------

Returns a list of all local host addresses for all network adapters.

**TCP.connect**(host, port, [cb])

<b>host</b>	String	Hostname or IP of server
<b>port</b>	Number	Port to connect to
<b>cb</b>	function(err, socket)	Optional callback

Connect to a TCP server socket.

**TCP.prototype.shutdown**([cb])

<b>cb</b>	function(err)	Optional callback
-----------	---------------	-------------------

Shutdown a TCP connection

**TCP.prototype.receive**(timeout, [cb])

<b>timeout</b>	Undefined or Number	Timeout duration in milliseconds or undefined for infinite
<b>cb</b>	function(err, buffer)	Optional callback

Receive data from a TCP socket. An empty buffer is returned if no data is received in the optional timeout interval. The value null is returned if the other side closed the connection while waiting.

**TCP.prototype.send**(buffer, [cb])

<b>buffer</b>	ArrayBuffer / Array / TypedArray	Data to write
<b>cb</b>	function(err)	Optional callback

Write data to a TCP socket.

**TCPServer.listen**(host, port, [cb])

<b>host</b>	String	Optional hostname or IP of client
<b>port</b>	Number	Port to listen on
<b>cb</b>	function(err, socket)	Optional callback

Create a TCP server object that listens on the given port number for incoming connections. A non null host argument will restrict the incoming connections to the given address.

**TCPServer.prototype.close**([cb])

<b>cb</b>	function(err)	Optional callback
-----------	---------------	-------------------

Close an open TCPServer.

**TCPServer.prototype.accept**( [cb])

<b>cb</b>	function(err, socket)	Optional callback
-----------	-----------------------	-------------------

Accept an incoming connection. Provides a TCP object as a socket response.

Creating a TCP server, waiting for a connection and sending "Hello World". Creating a TCP client and reading from the connection.

```
import { TCP, TCPServer } from "tcp.js"
import { Async } from "async.js"
const Port = 50872;
TCPServer.listen(null, Port).then(server => {
  Async.forever(cb => {
    server.accept().then(s => {
      cb();
      s.send(UTF8.encode("Hello World"));
    });
  });
});
```

```

    });
    return TCP.connect("localhost", Port);
}).then(client => {
    return client.receive();
}).then(data => {
    console.log(UTF8.decode(data));
});

```

## UDP Client Sockets – “udp.js”

**UDP.bind**(host, port, [cb])

<b>host</b>	String	Hostname or local server or null
<b>port</b>	Number	Port to bind with
<b>cb</b>	function(err, socket)	Optional callback

Create a UDP socket and bind it to the given local port

**UDP.prototype.connect**(host, port, [cb])

<b>host</b>	String	Hostname or IP of server
<b>port</b>	Number	Port to connect to
<b>cb</b>	function(err, socket)	Optional callback

Connect to a UDP server socket.

**UDP.prototype.shutdown**([cb])

<b>Cb</b>	function(err)	Optional callback
-----------	---------------	-------------------

Shutdown a UDP connection

**UDP.prototype.receive**([timeout, [cb]])

<b>timeout</b>	Undefined or Number	Timeout duration in milliseconds or undefined for infinite
<b>cb</b>	function(err, buffer)	Optional callback

Receive a datagram, from a UDP socket. The value null is returned if no data is received in the optional timeout interval

**UDP.prototype.send(buffer, [cb])**

<b>buffer</b>	ArrayBuffer / Array / TypedArray	Data to write
<b>cb</b>	function(err)	Optional callback

Write a datagram to a UDP socket.

## HTTP Client Requests – “http.js”

**Request.get(url, params, [cb])**

<b>url</b>	String	URL of the request
<b>params</b>	Object	Optional arguments for the request
<b>cb</b>	function(err, head, body)	Callback with the head and body of the result

Issue an http get request. The optional arguments are provided as an object:

<b>path</b>	String	Optional path component for the request
<b>query</b>	Object	Optional query parameters for the request
<b>header</b>	Object	Optional set of http headers

**Request.head(url, params, [cb])**

<b>url</b>	String	URL of the request
<b>params</b>	Object	Optional arguments for the request
<b>cb</b>	function(err, head, body)	Callback with the head of the result

Issue an http head request. The optional arguments are provided as an object (see get)

**Request.post(url, params, [cb])**

<b>url</b>	String	URL of the request
<b>params</b>	Object	Optional arguments for the request
<b>buffer</b>	ArrayBuffer	Body of the request

<b>cb</b>	function(err, head, body)	Callback with the head and body of the result
-----------	---------------------------	---

Issue an http post request. The optional arguments are provided as an object (see get)

**Request.put**(url, params, [cb])

<b>url</b>	String	URL of the request
<b>params</b>	Object	Optional arguments for the request
<b>buffer</b>	ArrayBuffer	Body of the request
<b>cb</b>	function(err, head, body)	Callback with the head and body of the result

Issue an http put request. The optional arguments are provided as an object (see get)

**Request.patch**(url, params, [cb])

<b>url</b>	String	URL of the request
<b>params</b>	Object	Optional arguments for the request
<b>buffer</b>	ArrayBuffer	Body of the request
<b>cb</b>	function(err, head, body)	Callback with the head and body of the result

Issue an http patch request. The optional arguments are provided as an object (see get)

**Request.delete**(url, params, [cb])

<b>url</b>	String	URL of the request
<b>params</b>	Object	Optional arguments for the request
<b>cb</b>	function(err, head, body)	Callback with the head of the result

Issue an http delete request. The optional arguments are provided as an object (see get)

**Request.authenticate**(host, user, password, [cb])

<b>host</b>	String	Host of the requests
<b>user</b>	String	Username
<b>password</b>	String	Password
<b>cb</b>	function(err)	Optional callback

Provide a user and password for the given host, to be used on further requests.

## HTTP Server “httpserver.js”

### HTTPServer(cb)

<b>cb</b>	function(request, response)	Server callback for incoming requests
-----------	-----------------------------	---------------------------------------

Create a http server object. The callback will be invoked with an **HTTPRequest** and **HTTPResponse** object and is expected to return a promise.

### async HTTPServer.prototype.listen(port)

<b>port</b>	Number	Port to listen for incoming connections
-------------	--------	---

Start listening on the incoming port.

### HTTPServer.prototype.close()

Close the http server for incoming connections

### HTTPRequest

<b>url</b>	String	Complete url of the request
<b>method</b>	String	Method of the request, such as “get” or “post”
<b>path</b>	String	Path component of the URL
<b>protocol</b>	String	Protocol HTTP 1.0 or 1.1
<b>headers</b>	Object	Object with the key value pairs of the request headers
<b>params</b>	Object	Object with the query params as key value pairs

### async HTTPRequest.prototype.read()

Read a chunk of data from the incoming request body. Returns null when all data has been read.

### async HTTPRequest.prototype.body()

Read the complete body of a request.



**HTTPResponse.prototype.setHeader(name, value)**

<b>name</b>	String	Name of the http header
<b>value</b>	String	Value for the header

Set one header parameter in the response

**HTTPResponse.prototype.setHead(status, headers)**

<b>status</b>	Number	Response status e.g. 200 for ok
<b>headers</b>	Object	Object with the header parameters

Set one header parameter in the response

**HTTPResponse.prototype.writeHead(status, headers)**

<b>status</b>	Number	Response status e.g. 200 for ok
<b>headers</b>	Object	Object with the header parameters

Write the response header, returns a promise for completion.

**HTTPResponse.prototype.sendHead()**

Send the previously set header, returns a promise for completion.

**HTTPResponse.prototype.write(chunk)**

<b>chunk</b>	ArrayBuffer, String, TypedArray	Chunk of response body
--------------	---------------------------------	------------------------

Write one chunk of response data. Sends the header if it was not sent before.

**HTTPResponse.prototype.end(chunk)**

<b>chunk</b>	ArrayBuffer, String, TypedArray	Body or final chunk of response body
--------------	---------------------------------	--------------------------------------

Write the body or final chunk of response data. Sends the header if it was not sent before.

Example for an http server that returns the query arguments as a json document.

```
import { HTTPServer } from "httpserver.js"

async function serve(req, res) {
  res.writeHead(200, {"content-type": "application/json"});
  res.end(req.params);
}

new HTTPServer(serve).listen(4080);
```

## Serial Port Communication – “serialport.js”

### SerialPort.list([cb])

<b>cb</b>	function(err)	Optional callback
-----------	---------------	-------------------

List the available serial ports of the system.

### SerialPort.open(device, rate, protocol, [cb])

<b>device</b>	String	Device name of the serial port to open
<b>rate</b>	Number	Baud rate for the connection
<b>protocol</b>	String	Optional protocol parameters
<b>cb</b>	function(err, socket)	Optional callback

Opens a serial port connection. The protocol string consists of four characters. The first denotes the handshake mechanism, X for software and H for hardware. The second character sets the byte length to either 7 or 8 bits. The third character specifies the number of stop bits and the final character sets and enables the parity check.

### SerialPort.prototype.close([cb])

<b>cb</b>	function(err)	Optional callback
-----------	---------------	-------------------

Shutdown a serial port connection

**SerialPort.prototype.receive**([cb])

<b>cb</b>	function(err, buffer)	Optional callback
-----------	-----------------------	-------------------

Receive data from a serial port connection.

**SerialPort.prototype.send**(buffer, [cb])

<b>buffer</b>	ArrayBuffer / Array / TypedArray	Data to write
<b>cb</b>	function(err)	Optional callback

Write data to a serial port connection.

## Invoking an external Process – “process.js”

**Process.create**(path, params, [cb])

<b>path</b>	String	Path to the process executable
<b>params</b>	Array of String	Command line parameters
<b>cb</b>	function(err, socket)	Optional callback

Create a process with the given path and command line arguments.

**Process.prototype.terminate**([cb])

<b>cb</b>	function(err)	Optional callback
-----------	---------------	-------------------

Terminate a process

**Process.prototype.read**([cb])

<b>cb</b>	function(err, buffer)	Optional callback
-----------	-----------------------	-------------------

Receive data from the standard output of a process

**Process.prototype.write**(buffer, [cb])

<b>buffer</b>	ArrayBuffer / Array / TypedArray	Data to write
<b>cb</b>	function(err)	Optional callback

Write data to the standard input of the process

**Process.prototype.wait**([cb])

<b>cb</b>	function(err)	Optional callback
-----------	---------------	-------------------

Wait for process completion

Dynamic link library plugin – “plugin.js”

**Plugin.open**(path, service, index, params, [cb])

<b>path</b>	String	Path to the library
<b>service</b>	String	Optional string parameter passed to the plugin open call. E.g. a secondary file name
<b>index</b>	Number	Optional index parameter
<b>params</b>	String	Optional configuration string
<b>cb</b>	function(err, socket)	Optional callback

Load a dynamic link library and call the exported open function with the additional parameters.

**Plugin.prototype.close**([cb])

<b>cb</b>	function(err)	Optional callback
-----------	---------------	-------------------

Close the handle to the plugin library

**Plugin.prototype.read**(size, [cb])

<b>size</b>	Number	Number of bytes to read
<b>cb</b>	function(err, buffer)	Optional callback

Receive data from the plugin library

**Plugin.prototype.write**(buffer, [cb])

<b>buffer</b>	ArrayBuffer / Array / TypedArray	Data to write
<b>cb</b>	function(err)	Optional callback

Write data to the plugin library

**Plugin.prototype.control**(command, buffer, [cb])

<b>command</b>	String	Command string passed to the plugin
<b>buffer</b>	ArrayBuffer / Array / TypedArray	Data to write
<b>cb</b>	function(err)	Optional callback

Call the exported control function of the plugin library and return the result buffer.

**Plugin.prototype.bind**(name, [cb])

<b>name</b>	String	Name of the function to bind to
<b>cb</b>	function(err)	Optional callback

Bind to a function of the plugin. The bound functions are called directly on the thread of the JSIQA execution block and may not wait or take a long time for processing. The signature of the exported function is provided by the exported bind function of the plugin.

## Mission, Block and Config Items – “mission.js”

### Block Configuration

**Mission.addConfig**(path, items, [cb])

<b>path</b>	String	Main path for the config items
<b>items</b>	Array / Object	Config items
<b>cb</b>	function(err, paths)	Optional callback

Add config items to the script config section. Each item may contain the following members:

<b>title</b>	String	User facing title of config item
<b>title2</b>	String	Alternate title for two state buttons
<b>name</b>	String	Internal unique name of config item

<b>short</b>	String	Short title for e.g. Ribbon
<b>icon</b>	String	Icon for buttons
<b>icon2</b>	String	Alternative icon for two state buttons
<b>type</b>	String	Type of config item (group, string, bool, button, number, float, integer, enum, colormap, frequencyProfile, color)
<b>value</b>	Number / String / Object	Start value of config item
<b>min</b>	Number	Minimum value of numeric config item
<b>max</b>	Number	Maximum value of numeric config item
<b>step</b>	Number	Step of numeric config item
<b>page</b>	Number	Page step
<b>pattern</b>	String, Array of Strings	Path / File pattern, names of enum values
<b>unit</b>	String	Unit of numeric config item
<b>titles</b>	Array of Strings	Titles of enum values
<b>mode</b>	String	Type dependent mode (file, newfile, path, button, slider, logslider)
<b>ephemeral</b>	Bool	The config item value is not retained between mission loads
<b>invisible</b>	Bool	The config item is not visible
<b>ribbonHeight</b>	Number	Required ribbon height (1 or 2)
<b>ribbonWidth</b>	Number	Required ribbon width (1, 2 or 3)
<b>ribbonPriority</b>	Number	Ribbon retainment priority -2 to 2
<b>ribbonTop</b>	Bool	Force the element into the top slot in the ribbon
<b>ribbonReorder</b>	Bool	Disable the ribbon reorder for layout if set to false
<b>ribbonPopUpSticky</b>	Bool	Keep the popup in the ribbon open when selected
<b>items</b>	Array	List of child items for a group

The following icons are available: play, pause, stop, left, right, up, down, radar, record, add, sub

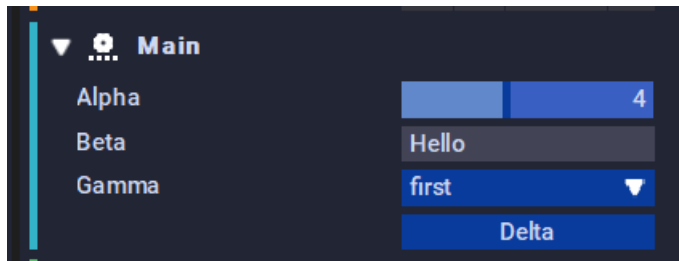
Creating for configuration items and moving them into the “Main” configuration area of the block.

```
import { Mission } from "mission.js"
var Config;
Mission.addConfig(null, {
  alpha: {title: "Alpha", type: "number", value: 4, mode: "slider",
    min: 0, max: 10, step: 1},
  beta: {title: "Beta", type: "string", value: "Hello"},
  gamma: {title: "Gamma", type: "enum", value: 0,
    pattern: ["first", "second", "third"]},
  delta: {title: "Delta", type: "button"}
}).then(c => {
  Config = c;
});
```

```

return Mission.mapConfig([
    {path: Config.alpha, target: "main"},
    {path: Config.beta, arget: "main"},
    {path: Config.gamma, target: "main"},
    {path: Config.delta, target: "main"},
]);
});

```



Configuration items are always created in the “Script Config” section of the block, but can be mapped into other areas of the configuration section.

**Mission.getConfig**(block, path, [cb])

<b>block</b>	String	Mission Block
<b>path</b>	String, Array, Object	Path of the config items
<b>cb</b>	function(err, paths)	Optional callback

Get the config value and settings of one or more config items. Using an object to specify the paths will return an object with the matching results.

```

return Mission.getConfig(null, {
    a: Config.alpha,
    b: Config.beta
});

```

The result in the debug view shows all members of the requested config items.

```

▼c
  ▼a
    .unit          ''
    .title         'Alpha'
    .value         4
    .visible       true
    .max           10
    .type          'number'
    .name          'alpha'
    .enabled       true
    .step          1
    .min           0
  ▼b
    .enabled       true
    .title         'Beta'
    .value         'Hello'
    .visible       true
    .type          'string'
    .name          'beta'

```

It is possible to query the configuration items of a different block in the mission. The “Block Graph Explorer” block provides an easy way to inspect the other blocks of the mission, and find the appropriate config items.



The explorer tables show the blocks and the available config items for each block:



title	name	type
Script	Script_0	Script
Block Graph Explorer	Block_Block Graph Explorer_f	Block Graph Explorer
<b>IQ Signal Generator</b>	<b>Block_IQSignalGenerator_0</b>	<b>IQSignalGenerator</b>

title	path	type	value	values
Play	/main/playbutton	bool	0	
Frequency Profile	/main/frequencyprofile	frequencyProfile	[]	
Adapt Center Frequency	/main/adaptcenter	bool	1	
Adapt Sample Rate	/main/adaptsamplerate	bool	1	
Generators	/main/generators	group		
Sweep	/main/generators/generator#iqsweepgenerator#4054961108	group		
Power	/main/generators/generator#iqsweepgenerator#4054961108/power	number	-15	
Start Frequency	/main/generators/generator#iqsweepgenerator#4054961108/startfreq	number	2.4e+09	
End Frequency	/main/generators/generator#iqsweepgenerator#4054961108/stopfreq	number	2.42e+09	
Direction	/main/generators/generator#iqsweepgenerator#4054961108/sweepmode	enum	Start Frequency	[]
Step Frequency	/main/generators/generator#iqsweepgenerator#4054961108/stepfreq	number	1e+06	
Offset Frequency	/main/generators/generator#iqsweepgenerator#4054961108/offsetfreq	number	1e+08	
Num Peaks	/main/generators/generator#iqsweepgenerator#4054961108/numpeaks	number	1	
Sweep Duration	/main/generators/generator#iqsweepgenerator#4054961108/sweepduration	number	10	
View	/view	group		
Background	/view/background			
<b>Unit</b>	<b>/view/powerunit</b>	<b>enum</b>	<b>dBm</b>	<b>[]</b>
Font Size	/view/fontsize	number	10	
Hide Watermark	/view/hidewatermark	bool	0	

```
return Mission.getConfig("Block_IQSignalGenerator_0", {
    rate: "main/samplerate"
});
```

**Mission.setConfig(block, values, [cb])**

<b>block</b>	String	Mission Block
<b>values</b>	Array	List of "path", "value", "enabled" and "pattern" item objects
<b>cb</b>	function(err, paths)	Optional callback

Set the config items to the given values.

**Mission.mapConfig(mappings, [cb])**

<b>mappings</b>	Array	List of "path", "target" and "ribbon" item objects
<b>cb</b>	function(err, paths)	Optional callback

Maps the config items with given the path to the new target location.

### Mission.chainConfig(config, source, [cb])

<b>config</b>	String	Path to target config item
<b>source</b>	String	Path to source config item
<b>cb</b>	function(err, paths)	Optional callback

Chains a source config item to a target config item. The source item will then mirror the target item changes.

### Mission.listenConfig(cb)

<b>cb</b>	function(path, value, user)	Callback for config item changes
-----------	-----------------------------	----------------------------------

Add a callback for configuration changes.

### Mission.dialog(title, items, [cb])

<b>Title</b>	String	Title of the dialog
<b>items</b>	Array / Object	Config items
<b>cb</b>	function(err, paths)	Optional callback

Display a modal dialog with the given list of config items, and return an object/array of the results if the user accepts the dialog – null, if the dialog is canceled.

### Mission.getFrequencyProfile(id, [cb])

<b>id</b>	String	ID of the frequency profile
<b>cb</b>	function(err, paths)	Optional callback

Return a recursive object of the frequency profile.

<b>start</b>	Number	Start frequency
<b>stop</b>	Number	End frequency
<b>id</b>	String	ID of the profile
<b>title</b>	String	Title of the profile
<b>children</b>	Array	Array of child frequency profiles
<b>channels</b>	Array	List of individual channels in the profile
<b>ranges</b>	Array	List of ranges of channels

### Graph Messages

Graph messages are sent from one block to all blocks up and/or downstream. A specific receiver can be selected using its name or UUID.

### Mission.receiveGraphMessages(cb)

<b>cb</b>	function(msg)	Callback for graph messages
-----------	---------------	-----------------------------

Receive graph messages that are sent by other blocks.

Graph message objects have the following fields (not all fields are present or valid for all messages):

<b>type</b>	String	Type of graph message, see below for a list of available message type names
<b>startFrequency</b>	Number	Start frequency
<b>endFrequency</b>	Number	End frequency
<b>stepFrequency</b>	Number	Step frequency or sample rate
<b>rbwFrequency</b>	Number	RBW Frequency
<b>offsetFrequency</b>	Number	Frequency relative to center
<b>startTime</b>	Number	Start time in seconds since the epoch
<b>endTime</b>	Number	End time in seconds since the epoch
<b>poiDuration</b>	Number	Requested 100% POI detection duration
<b>startPower</b>	Number	Lower power range limit
<b>endPower</b>	Number	Upper power range limit
<b>sampleSize</b>	Number	Size of a sample
<b>attenuation</b>	Number	Signal attenuation in dB
<b>shortPulse</b>	String	Short pulse modes time/frequency/default
<b>antennaSegment</b>	Number	Index of the segment in a multi segment antenna
<b>latitude</b>	Number	Geo location latitude
<b>longitude</b>	Number	Geo location longitude
<b>altitude</b>	Number	Geo location altitude / elevation
<b>referenceLevel</b>	Number	Reference level for a measurement in dBm
<b>receiverUUID</b>	String	Optional UUID of the receiver block
<b>receiverName</b>	String	Optional name of the receiver block
<b>azimuth</b>	Number	Geo location/direction azimuth
<b>declination</b>	Number	Geo location/direction declination
<b>aperture</b>	Number	Aperture of a camera
<b>frequencyProfile</b>	Array, String	Name of frequency profile
<b>mode</b>	Number	Mode configuration
<b>pattern</b>	Array of Number	Sequence of numbers representing a pattern of e.g. frequency jumps
<b>upstream</b>	Bool	Send the message upstream if true
<b>downstream</b>	Bool	Send the message downstream if true
<b>mirror</b>	Bool	Mirror the message at the end of a chain
<b>global</b>	Bool	Send the message to all blocks
<b>antennaUUID</b>	String	UUID of the antenna
<b>filename</b>	String	Filename for load or save of data
<b>streamname</b>	String	Name of the stream
<b>config</b>	Object	Config Items

The following message types are supported:

- **CURSOR\_CHANGED** The cursor has changed
- **CENTER\_AT\_FREQUENCY** Center the receiver at the frequency
- **SET\_FREQUENCY\_RANGE** Set the frequency range
- **SET\_TIME\_RANGE** Set the time range
- **REPLAY\_TIME\_RANGE** Replay the given time range
- **REPLAY\_STEADY\_RANGE** Replay the given range in steady mode
- **ANTENNA\_SEGMENT\_CHANGED** The antenna segment has changed
- **ANTENNA\_CHANGED** The antenna has changed
- **PASS\_TOKEN**
- **STREAM\_STARTING** The stream is starting
- **STREAM\_STOPPED** The stream has stopped
- **HF\_PATH\_CHANGED** The HF path has changed
- **CHANGE\_ANTENNA\_POS** Change the antenna position
- **REQUEST\_ANTENNA** Request an antenna changed message
- **COLLECT\_CONFIG** Collect configuration data
- **GET\_CONFIG** Get configuration data
- **SET\_CONFIG** Set configuration data
- **NOTIFY\_FREQUENCY\_PROFILE** Frequency profile has changed
- **START\_STREAMING** Start streaming
- **STOP\_STREAMING** Stop streaming
- **NETWORK\_CONNECT** Connect to an upstream network node
- **NETWORK\_DISCONNECT** Disconnect an upstream network node
- **START\_ROTATION** Start antenna rotation
- **STOP\_ROTATION** Stop antenna rotation
- **LOCATION\_CHANGED** Location as changed
- **PROCESSING\_CHANGED** Processing has changed
- **COLLECT\_FEATURES** Collect upstream features
- **START\_GENERATOR** Start signal generator
- **STOP\_GENERATOR** Stop signal generator
- **SET\_GENERATOR\_FREQUENCY** Set frequency of signal generator
- **START\_RECORDING** Start recording
- **STOP\_RECORDING** Stop recording
- **JAMMER\_ACTIVATE** Activate a jammer
- **JAMMER\_DEACTIVATE** Deactivate a jammer
- **JAMMER\_POWER\_ON** Turn jammer power on
- **JAMMER\_POWER\_OFF** Turn jammer power off
- **CAMERA\_START** Start a camera
- **CAMERA\_STOP** Stop a camera
- **CAMERA\_TARGET** Set camera target
- **CAMERA\_WIPER** Clean camera using wiper
- **REQUEST\_HEALTH\_STATUS** Request health status

- **HEALTH\_STATUS** Health status notification
- **REQUEST\_REMOTE\_CONFIG** Request remote config items
- **REMOTE\_CONFIG** Remote config item notification
- **SET\_REMOTE\_CONFIG** Set remote config items
- **REMOTE\_SAVE\_MISSION** Save remote mission
- **REMOTE\_RELOAD\_MISSION** Reload remote mission
- **ROUTE\_STREAM** Route stream through a multiplexer
- **SET\_HIGHLIGHT** Set highlight sector

**Mission.sendGraphMessage** (msg, cb)

<b>msg</b>	Object	Graph message object
<b>cb</b>	function(err, msg)	Optional callback

Send a graph message up- or downstream through the graph. Returns the response message with the optional callback.

```
Mission.sendGraphMessage({
  upstream: true,
  type: "START_STREAMING"
});
```

**Mission.sendGraphMessageBy** (con, msg, cb)

<b>con</b>	String	Connector name to send from
<b>msg</b>	Object	Graph message object
<b>cb</b>	function(err, msg)	Optional callback

Send a graph message up- or downstream through the graph. Returns the response message with the optional callback.

**Mission.realtimeGraphMessenger** (cb)

<b>cb</b>	function(err, msg)	Optional callback
-----------	--------------------	-------------------

Creates a realtime graph messenger function that can be used to send graph message on the script block thread. The function accepts a message and an optional timeout in milliseconds to wait for the global graph mutex.

```
var messenger = await Mission.realtimeGraphMessenger();

messenger ({
```

```

upstream: true,
type: "SET_FREQUENCY_RANGE"
startFrequency: 1000.0e6,
endFrequency: 1100.0e6,
}, 1000);

```

### Mission.filterGraphMessages (msgs, cb)

<b>msgs</b>	Array of Strings	Graph message types to filter
<b>cb</b>	function(err, msg)	Optional callback

Blocks the up- and downstream graph messages with the given types from propagating through the block to other blocks of the graph.

## Logging

### Mission.logInfoCenter(cb)

<b>msg</b>	String or Array of Strings	Messages to log
<b>cb</b>	function(err, msg)	Optional callback

Log messages at info level in the info center

### Mission.logInfoCenterWarning(cb)

<b>msg</b>	String or Array of Strings	Messages to log
<b>cb</b>	function(err, msg)	Optional callback

Log messages at warning level in the info center

### Mission.logInfoCenterError(cb)

<b>msg</b>	String or Array of Strings	Messages to log
<b>cb</b>	function(err, msg)	Optional callback

Log messages at error level in the info center

## Graph and Block management

### Mission.block([cb])

<b>cb</b>	function(err, block)	Optional callback
-----------	----------------------	-------------------

Returns the info of the current block, and additional global information:

<b>mission</b>	String	Path and filename of the mission
<b>cdata</b>	String	Path to fixed program data
<b>gdata</b>	String	Path to common program data
<b>udata</b>	String	Path to user specific data
<b>ddata</b>	String	Path to user documents
<b>oskernel</b>	String	Name of the os kernel
<b>osversion</b>	String	OS version string
<b>osproduct</b>	String	Name of the os product
<b>inputs</b>	Array	Array of inputs
<b>outputs</b>	Array	Array of outputs

### Mission.blocks([cb])

<b>cb</b>	function(err, blocks)	Optional callback
-----------	-----------------------	-------------------

Returns an array with the structure of all blocks in the mission.

<b>uuid</b>	String	UUID of the block
<b>name</b>	String	Name of the block
<b>title</b>	String	Title of the block
<b>description</b>	String	Description of the block
<b>inputs</b>	Array	Array of inputs
<b>outputs</b>	Array	Array of outputs

The connection objects in the inputs and outputs array contain the following information:

<b>name</b>	String	Name of the connector
<b>title</b>	String	Title of the connector
<b>type</b>	String	Type of the connector
<b>output</b>	Object	Name of block and connector of the output that is connected to this input
<b>inputs</b>	Array	Array of name and block of the connected inputs

### Mission.listenGraph (cb)

<b>cb</b>	function(msg)	Callback for graph changed messages such as connect or disconnect
-----------	---------------	---

### Mission.setConnectors(cons, [cb])

<b>cons</b>	Object of Object	Object of connector objects to change.
-------------	------------------	--

Change the dynamic connectors of a block (“in4” .. “in19” and “out4” .. “out19”). Each element of the cons parameter is one output, selected by its name.

<b>visible</b>	Boolean	Visibility of the connector
<b>title</b>	String	Title of the connector
<b>type</b>	String	Type of the connector

### Mission.setHealthState(state, [config], [cb])

<b>state</b>	String	Name of the current module health state: unknown, idle, booting, ready, starting, operational, running, warning, critical
<b>config</b>	Object	Config item object with the health state
<b>cb</b>	function(err)	Optional callback

Set the current script health state. The “Show Health Status” checkbox has to be checked for this to show up in the block graph.

### Mission.getMarkers(block, [cb] )

<b>block</b>	String	Name of block
<b>cb</b>	function(err, msg)	Optional callback

Get a list of the markers of the given block with x, y and z measurement.

## Charts and User Interface

### Mission.initGrid(columns, rows, [cb])

<b>columns</b>	Number	Number of columns
<b>rows</b>	Number	Number of rows
<b>cb</b>	function(err)	Optional callback

Initialize the view grid of the script block to the number of columns and rows

### Mission.onkey(cb)



<b>cb</b>	function(event)	Callback
-----------	-----------------	----------

Add callback for key input events

**Mission. onmouse(cb)**

<b>cb</b>	function(event)	Callback
-----------	-----------------	----------

Add callback for mouse input events

**Mission.addChart(name, type, x, y, width, height, params, [cb])**

<b>name</b>	String	Name of the chart element
<b>type</b>	String	Type of the chart element
<b>x</b>	Number	Starting column
<b>y</b>	Number	Starting row
<b>width</b>	Number	Number of columns to cover
<b>height</b>	Number	Number of rows to cover
<b>params</b>	Object	Parameters of the chart element
<b>cb</b>	function(err, chart)	Optional callback

Create a chart element in the view grid. The following chart elements and matching parameters are available.

<b>padding</b>	Number	Padding around chart element
<b>show</b>	Bool	Show/Hide the element
<b>priority</b>	Number	Depth priority of elements. Higher priority elements will be drawn in front of lower priority elements in the same grid cell
<b>mousemove</b>	Bool	Transmit mouse movements with the mouse event callback

**statictext** – static text element

<b>text</b>	String	Text to display
<b>fscale</b>	Number	Relative scale factor to the user selected font size
<b>color</b>	Object / String	Text color ( an object with r, g, b elements)
<b>background</b>	Object / String	Background color
<b>hover</b>	Object / String	Hover color

<b>click</b>	Object / String	Click color
<b>hoverchecked</b>	Object / String	Hover and checked color
<b>checked</b>	Object / String	Checked color
<b>align</b>	String	Horizontal alignment (left, right, center)
<b>valign</b>	String	Vertical alignment (top, bottom, center)
<b>rounded</b>	Number	Radius of rounded corners
<b>input</b>	String	Config item that is triggered by this text field

#### grid – sub grid

<b>width</b>	Number	Number of grid columns
<b>height</b>	Number	Number of grid rows
<b>background</b>	Object / String	Background color ( an object with r, g, b elements)
<b>relayout</b>	Bool	Force a re-layout of the grid if the visibility of its elements change

#### spacer – spacer element for layout

<b>width</b>	Number	Requested width in pixel
<b>height</b>	Number	Requested height in pixel
<b>minWidth</b>	Number	Minimum width in pixel
<b>minHeight</b>	Number	Minimum height in pixel
<b>maxWidth</b>	Number	Maximum width in pixel
<b>maxHeight</b>	Number	Maximum height in pixel
<b>expandWidth</b>	Number	Usable width in pixel
<b>expandHeight</b>	Number	Usable height in pixel
<b>background</b>	Object / String	Background color ( an object with r, g, b elements)

#### widget – interactive element based on config item

<b>config</b>	String	Configuration element
---------------	--------	-----------------------

#### texttable – table of text elements

<b>columns</b>	Array of Strings	Names of the table columns
	Array of Objects	Name (name) and alignment (align) data for the table columns
<b>cells</b>	Array	List of the table cell entries. This can either be a one-dimensional array for all cells, or an array of arrays or an array of objects
<b>pattern</b>	String	Reference string for the cells

<b>inputRowIndex</b>	String	Config item that receives the currently selected table row
<b>inputColumnName</b>	String	Config item that receives the name of the currently selected table config
<b>rows</b>	Array of Objects	Background (background) and text color (color) values for each row of the table
<b>selected</b>	Number	Index of the row to select/highlight or -1 for none
<b>focus</b>	Object	Clear (null) or set the focus with an object with row and column fields.
<b>cell</b>	Object	Changing a single cell, providing the row, column and value

Data elements:

	Array	List of the table cell entries. This can either be a one-dimensional array for all cells, or an array of arrays or an array of objects
--	-------	--

**statuslist** – list of textual name value pairs

<b>values</b>	Object	Object with the name value pairs
<b>fields</b>	Array of string	List of the names of the entries

**statustimechart** – simple xy chart of timed measurements

<b>unit</b>	String	Unit of measurement
<b>min</b>	Number	Minimum value
<b>max</b>	Number	Maximum value
<b>scale</b>	Number	Scale factor
<b>duration</b>	Number	Duration of measurement in seconds

Data elements:

<b>time</b>	Number	Time of measurement
<b>value</b>	Number	Value of measurement

**statusxychart** – simple xy chart of 2d measurements

<b>unitX</b>	String	Unit of measurement
<b>minX</b>	Number	Minimum value
<b>maxX</b>	Number	Maximum value

<b>scaleX</b>	Number	Scale factor
<b>unitY</b>	String	Unit of measurement
<b>minY</b>	Number	Minimum value
<b>maxY</b>	Number	Maximum value
<b>scaleY</b>	Number	Scale factor

Data elements:

<b>x</b>	Number	X Value
<b>y</b>	Number	Y Value

**xychart** – xy chart of 2d measurements

<b>unitX</b>	String	Unit of measurement
<b>minX</b>	Number	Minimum value
<b>maxX</b>	Number	Maximum value
<b>unitY</b>	String	Unit of measurement
<b>minY</b>	Number	Minimum value
<b>maxY</b>	Number	Maximum value
<b>cursorX</b>	Number	Current cursor X position
<b>cursorY</b>	Number	Current cursor Y position
<b>markX</b>	Number	Current mark X position
<b>markY</b>	Number	Current mark Y position
<b>cursor</b>	Boolean	Is cursor visible
<b>items</b>	Array of Objects	Array of data elements (empty array to clear chart)

Data elements:

<b>name</b>	String	Name of the data set
<b>title</b>	String	Optional title displayed
<b>info</b>	String	Optional info for tooltips
<b>priority</b>	Number	Priority for z ordering
<b>lineColor</b>	Object / String	Color of line elements
<b>markColor</b>	Object / String	Color of mark elements
<b>lineWidth</b>	Number	Width of the lines connecting the values
<b>markWidth</b>	Number	Width of the marks at the values
<b>values</b>	Array of Objects	Array of the values comprised of objects with x and y elements.

**waterfall** – waterfall view of spectrum data

<b>source</b>	String	Name of the source DSP block providing the data
<b>output</b>	String	Output of the source block

<b>colormap</b>	String	Name of the colormap config item
<b>compressor</b>	String	Name of an optional time compressor block
<b>inverttime</b>	Bool	Invert vertical orientation

**histogram** – histogram view of spectrum data

<b>source</b>	String	Name of the source DSP block providing the data
<b>output</b>	String	Output of the source block
<b>colormap</b>	String	Name of the colormap config item
<b>interpolation</b>	Bool	Interpolate the values between the bins of the histogram
<b>unit</b>	String	Unit of the bin values, either “percentage” for a histogram or “dbm” for multi spectral

**spectrum** – spectrum view of spectrum data

<b>source</b>	String	Name of the source DSP block providing the data
<b>output</b>	String	Output of the source block
<b>source2</b>	String	Name of the second optional source DSP block providing the data
<b>output2</b>	String	Output of the second source block
<b>colormap</b>	String	Name of the colormap config item
<b>legend</b>	String	Name of the trace legend config item
<b>traces</b>	Array of Strings / Objects	List of the traces to be displayed in the chart, any of “average”, “maxhold”, “minhold”, “quasimaxhold”, “maxfall”, “outline”, “pulsedshape”, “histogram”, “samplehold”, “clearwrite”
<b>markers</b>	Array of Objects	

Traces Objects

<b>title</b>	String	User facing name of the trace
<b>type</b>	String	Type of the trace (see above)
<b>name</b>	String	Application facing name of the trace
<b>ephemeral</b>	Bool	Do not store or remember the configuration of this trace
<b>config</b>	Object	Base configuration of the trace

## Markers Objects

<b>title</b>	String	User facing name of the trace
<b>source</b>	String	Name of the source trace for the marker
<b>name</b>	String	Application facing name of the marker
<b>ephemeral</b>	Bool	Do not store or remember the configuration of this marker
<b>config</b>	Object	Base configuration of the marker

## Marker Configuration

<b>color</b>	Object or String	Color of symbol/bar
<b>textcolor</b>	Object or String	Color of text
<b>showname</b>	Bool	Show the name
<b>showindex</b>	Bool	Show the index
<b>showxpos</b>	Bool	Show the horizontal value
<b>showypos</b>	Bool	Show the vertical value
<b>showzpos</b>	Bool	Show additional value
<b>showfunct</b>	Bool	Show the function value
<b>rotatetext</b>	Bool	Rotate the text in dense settings
<b>msize</b>	Number	Size of the symbol
<b>msymbolv</b>	String	Marker shape for value/value, can be "None", "Circle", "X-Cross", "Circle X", "Cross", "Box", "Disk", "Quad" or "Triangle"
<b>msymbolvr</b>	String	Marker shape for value/range, can be "None", "Lines", "Bar" or "Flags"
<b>msymbolvf</b>	String	Marker shape for value, can be "None", "Lines", "Dotted"
<b>msymvolrr</b>	String	Marker shape for range/range, can be "None", "Box", "Border", "Corners"
<b>msymbolrf</b>	String	Marker shape for range, can be "None", "Box", "Border", "Corners" or "Flags"
<b>showxline</b>	Bool	Show a line for the horizontal location
<b>showyline</b>	Bool	Show a line for the vertical location
<b>showzline</b>	Bool	Show a line for the additional location
<b>mode</b>	String	Mode can be "Samples", "Frequencies", "Bands", "Peaks"
<b>accumulate</b>	String	Accumulation mode/function, can be "Minimum", "Maximum", "Average", "Sum" or "Ch.Power", "Spectral Density" or "Spike"
<b>sortorderpeaks</b>	String	Sort order in "Peaks" mode, either "Power", "Frequency" or "Time"
<b>sourcepeaks</b>	String	Name of the source trace
<b>name</b>	String	Name of the markers
<b>count</b>	Number	Number of markers in the series
<b>symmetric</b>	Bool	Place markers symmetric around center

<b>harmonics</b>	Bool	Place markers in harmonic series
<b>modefreq</b>	String	Frequency mode, "RTBW", "Center", "Area", "Custom"
<b>basefreq</b>	Number	Base frequency
<b>stepfreq</b>	Number	Step frequency in series
<b>spanfreq</b>	Number	Frequency width in band mode
<b>deltafreq</b>	Number	Frequency offset in depended markers
<b>excursion</b>	Number	Excursion in peak mode
<b>threshold</b>	Number	Threshold in peak mode
<b>separation</b>	Number	Separation in peak mode

#### Parameters

<b>markers</b>	Object of Objects	Configuration objects for the markers
<b>traces</b>	Object of Objects	Configuration objects for the markers
<b>stream</b>	Object	Meta information for the incoming sample stream

#### timeshift – timeshift view

<b>source</b>	String	Name of the source DSP block providing the data
<b>colormap</b>	String	Name of the colormap config item

#### geomap – geographic map

<b>source</b>	String	Name of the source DSP block providing the location
<b>output</b>	String	Name of the output of the source DSP block
<b>colormap</b>	String	Name of the colormap config item
<b>latitude</b>	Number	Latitude of map center
<b>longitude</b>	Number	Longitude of map center
<b>markers</b>	Array of Objects	List of marker objects to show on maps

Marker objects may contain the following elements

<b>name</b>	String	Name of the marker
<b>latitude</b>	Number	Latitude of marker
<b>longitude</b>	Number	Longitude of marker
<b>azimuth</b>	Number	Direction of marker

<b>shape</b>	String	Shape of the marker, any of “circle”, “cross”, “arrow”, “beam”, “line”, “arrowcircle”, “compassarrow”, “triangle”, “quad”, “pentagon”, “hexagon”, “octagon”, “star3”, “star4”, “star5”, “star6”, “halfcircle”, “path”, “tripath”
<b>radius</b>	Number	Radius of marker in meter
<b>size</b>	Number	Radius of marker in pixel
<b>width</b>	Number	Width of radius outline in pixel
<b>color</b>	Object / String	Outline color of marker
<b>fill</b>	Object / String	Fill color of marker
<b>path</b>	Array	Array of objects with latitude and longitude for path markers

**canvas2d** – two dimensional canvas element

<b>width</b>	Number	Expected width in pixel
<b>height</b>	Number	Expected height in pixel
<b>aspectRatio</b>	String	Aspect ratio control if the element has a different than expected size, can be “free”, “width”, “height”, “stretch”, “shrink” or “clipped”
<b>items</b>	Array of Objects	Array of items to draw

Items can be provided by the construction parameter, using setParams or with addData. Each item is identified by its unique name.

<b>name</b>	String	Name of the data set
<b>title</b>	String	Optional title displayed
<b>info</b>	String	Optional info for tooltips
<b>priority</b>	Number	Priority for z ordering
<b>alignX</b>	Number	Alignment of the title relative to the first coordinate, can be 0 to 1
<b>alignY</b>	Number	Alignment of the title relative to the first coordinate, can be 0 to 1
<b>offsetX</b>	Number	X offset for coordinates of this item and all its child items
<b>offsetY</b>	Number	Y offset for coordinates of this item and all its child items
<b>scale</b>	Number	Scale factor for coordinates of this item and all its child items
<b>lineColor</b>	Object / String	Color of line elements
<b>markColor</b>	Object / String	Color of mark elements
<b>fillColor</b>	Object / String	Color or gradient for filling
<b>lineWidth</b>	Number	Width of the lines connecting the values



<b>markWidth</b>	Number	Width of the marks at the values
<b>textScale</b>	Number	Size of text relative to default, must be in the range of 0.1 to 2.0
<b>fill</b>	Bool	Fill the polygon
<b>close</b>	Bool	Close the lines to form a polygon
<b>coords</b>	Array of Objects	Array of the coordinates comprised of objects with x and y elements.
<b>items</b>	Array of Objects	Array of child items of this item
<b>input</b>	String	Path to optional input element controlled by this canvas item
<b>inputValue</b>	Number	Value index for enum input items
<b>hoverColor</b>	Object / String	Alternate fill color when hovering the mouse over the item
<b>pushColor</b>	Object / String	Alternate fill color when pushing the left button over the item
<b>checkedColor</b>	Object / String	Alternate fill color when the input item associated to the item is checked
<b>checkedHoverColor</b>	Object / String	Alternate fill color when hovering the mouse over the item if checked

```
let canvas = await Mission.addChart("Area", "canvas2d", 0, 0, 1, 1, {items: [
  {name: "tri",
    coords: [{x: 80, y: 20}, {x: 140, y: 140}, {x: 20, y: 140}],
    close: true, fill: true, lineColor: "#ffffff"},
  {name: "title", title: "Triangle",
    coords: [{x: 80, y: 141}], alignX: 0.5}
]});
```

A gradient has the following structure:

<b>colors</b>	Array	Array of color objects
<b>x0</b>	Number	Horizontal coordinate of gradient start
<b>y0</b>	Number	Vertical coordinate of gradient start
<b>x1</b>	Number	Horizontal coordinate of gradient stop
<b>y1</b>	Number	Vertical coordinate of gradient stop

**hsplitter** – horizontal splitter

**vsplitter** – vertical splitter

**Chart.prototype.setParams**(params, cb)

<b>params</b>	Object	Object containing the named parameters
<b>cb</b>	function(err)	Optional callback

Set parameters for an existing chart

**Chart.prototype.getParams**(params, cb)

<b>params</b>	Array	Array of parameter names
<b>cb</b>	function(err)	Optional callback

Get parameters for an existing chart

Common parameters to get:

<b>mouseX</b>	Number	Current mouse position in pixel
<b>mouseY</b>	Number	Current mouse position in pixel
<b>mouseLatitude</b>	Number	Mouse geo location
<b>mouseLongitude</b>	Number	Mouse geo location
<b>cursor</b>	Bool	Chart cursor is visible
<b>cursorX</b>	Number	Chart cursor position
<b>cursorY</b>	Number	Chart cursor position
<b>markX</b>	Number	Chart cursor/mark position
<b>markY</b>	Number	Chart cursor/mark position
<b>selected</b>	Number	Current selected table row

**Chart.prototype.addData**(data, cb)

<b>data</b>	Array or Object	Data to add to the chart
<b>cb</b>	function(err)	Optional callback

Add data to a chart.

**Chart.prototype.remove**(params, cb)

<b>cb</b>	function(err)	Optional callback
-----------	---------------	-------------------

Remove a chart from the grid

**Chart.prototype.show**(show, cb)

<b>show</b>	Boolean	Show or Hide
<b>cb</b>	function(err)	Optional callback

Show or hide an existing chart

**Chart.prototype.addChart**(name, type, x, y, width, height, params, [cb])

<b>name</b>	String	Name of the new chart element
<b>type</b>	String	Type of the new chart element
<b>x</b>	Number	Starting column
<b>y</b>	Number	Starting row
<b>width</b>	Number	Number of columns to cover
<b>height</b>	Number	Number of rows to cover
<b>params</b>	Object	Parameters of the chart element
<b>cb</b>	function(err, chart)	Optional callback

Add additional chart elements in the grid of an existing chart element

Streaming of IQ and other Measurement data – “dspstream.js”

**DSPStream.sendPacket**(stream, packet, cb)

<b>stream</b>	Number	Stream index 0..3
<b>packet</b>	Stream Packet Object	Packet to send
<b>cb</b>	function(err)	Optional callback

Send a packet to the dsp block output with index “stream”.

The packet should / may contain the following members:

<b>size</b>	Number	Size of a single sample
<b>depth</b>	Number	Depth of a single sample (e.g. bins in a histogram)
<b>payload</b>	String	Name of the payload type
<b>unit</b>	String	Name of the sample units
<b>startTime</b>	Number	Start time in seconds since the epoch
<b>endTime</b>	Number	End Time in seconds since the epoch
<b>startFrequency</b>	Number	Start of frequency range
<b>endFrequency</b>	Number	End of frequency range
<b>stepFrequency</b>	Number	IQ Sample Rate / Size of FFT Spectrum bin
<b>minValue</b>	Number	Minimum value in range
<b>maxValue</b>	Number	Maximum value in range

<b>start</b>	Boolean	Start of segment indicator
<b>end</b>	Boolean	End of segment indicator
<b>antenna</b>	Antenna Object	Antenna Information
<b>categories</b>	Array	List of category names
<b>samples</b>	Array / TypedArray / Object	Sample data

Spectrum samples should be provided as an array of typed arrays, IQ data as a typed array of complex values.

The antenna object has the following members:

<b>uuid</b>	String	UUID of the antenna
<b>latitude</b>	Number	Latitude of the antenna
<b>longitude</b>	Number	Longitude of the antenna
<b>azimuth</b>	Number	Azimuth of the antenna
<b>declination</b>	Number	Declination of the antenna
<b>segments</b>	Array	Array of antenna segments

**DSPStream.receivePackets**(stream, cb)

<b>stream</b>	Number	Stream index 0..3
<b>cb</b>	function(flags, meta, [samples])	Optional callback

Install a callback to be invoked when a stream packet is received by the dsp block input with index “stream”. The callback function can receive the following arguments:

<b>flags</b>	Number	Name of the payload type
<b>meta</b>	Object	Name of the sample units
<b>samples</b>	function([index])	Start time in seconds since the epoch

The meta data element receives an object similar to the object required for the sendPacket function. The samples can be retrieved by calling the samples function with an integer index, or no index for the complete data.

**DSPStream.streamTimer**(cb)

<b>cb</b>	function(err, data)	Optional callback
-----------	---------------------	-------------------

Create a stream timer. Returns a function that provides the high precision stream time of the DSP graph.

**DSPStream.addBlocks**(blocks, cb)

<b>blocks</b>	Array / Object	Block specifications
<b>cb</b>	function(err, data)	Optional callback

Install a series of dsp processing block into the dsp processing child graph of the script block.

The block specifications are provided as either an array or an object of the following objects.

<b>type</b>	String	Type of DSP block
<b>name</b>	String	Optional name of the DSP block
<b>config</b>	Array	Optional array of path / value pairs for initial configuration

**DSPStream.connectBlocks(connections, cb)**

<b>connections</b>	Array	Connection specifications
<b>cb</b>	function(err, data)	Optional callback

Connect a series of blocks in the dsp child graph of the script block. Each connection specification is an object with the following members:

<b>source</b>	String	Name of source DSP Block
<b>output</b>	String	Optional name of the DSP output
<b>drain</b>	String	Name of target DSP Block
<b>input</b>	String	Optional name of the DSP input

The input DSP blocks for the inbound connections are named “in0”, “in1”, “in2” and “in3” the names of the outbound connections are “out0”, “out1”, “out2” and “out3”. The script DSP block name is “script” and it has the inputs “in0”, “in1”, “in2” and “in3” and the outputs “out0”, “out1”, “out2” and “out3”.

The inbound connectors are connected to the input connections of the DSP script block and the output connectors of the DSP script block are connected to the outbound connections. You may freely reroute this connections.

## Plugin Dynamic Link Library

Dynamic link libraries can be used to extend the functionality of the JSIQA system. Communication with the library can be implemented with asynchronous invocation using function call and a callback function or synchronous with direction function calls and an immediate return value.

One type of callback function is used for all asynchronous library calls:

```
typedef void (* PluginCalllback) (  
    void * context,  
    int64_t data,  
    const void * buffer);
```

The callback function and the context parameter are provided by the caller of the library function, the data value and the buffer are the function return. A negative value for data designates an error.

The library should export the following functions:

```
void RTSA_PluginOpen (
    const char16_t * name,
    int64_t index,
    const char16_t * params,
    PluginCallback callback,
    void * context);
```

Called initially after loading the library. The name, index and params value may be used by the library to provide different resources. The returned value is used as a handle for all further calls.

```
void RTSA_PluginClose (
    int64_t handle,
    PluginCallback callback,
    void * context);
```

Called when the plugin is closed by the javascript code. There is no guarantee that this function will be called (e.g. it will not be called if the object representing the handle is destroyed by the garbage collector). Resource management must therefore also be handled by the terminate function.

```
void RTSA_PluginTerminate (
    int64_t handle,
    PluginCallback callback,
    void * context);
```

Called before the plugin is unloaded. The plugin may not call any other pending plugins or access buffers after completing this call by calling the callback function. This function will be called regardless of the way the javascript object is destroyed.

```
void RTSA_PluginRead (
    int64_t handle,
    int64_t size,
    PluginCallback callback,
    void * context);
```

Reads an amount of binary data from the library into a javascript ArrayBuffer. The library may use the provided size parameter as an indication for the requested amount. The buffer provided with the callback remains in the possession of the library.

```
void RTSA_PluginWrite (
    int64_t handle,
    int64_t size,
    const void * buffer,
    PluginCallback callback,
    void * context);
```

Write an amount of binary data from the javascript code to the library. The buffer pointer remains valid until the library is closed or the call is completed using the close function. The amount of data written can be signaled using the data parameter of the callback.

```

void RTSA_PluginControl(
    int64_t handle,
    const char16_t * name,
    int64_t size,
    const void * buffer,
    PluginCallback callback,
    void * context);

```

Invokes a control function in the library. The function is denoted by the name string. Input data is provided using the binary buffer and returned using a buffer with the callback.

```

void RTSA_PluginBind(
    int64_t handle,
    const char16_t * name,
    PluginCallback callback,
    void * context);

```

Bind a synchronous library function by name to a javascript function. The function pointer is provided with the data portion of the callback, the signature with a unicode string in the buffer.

The general signature of the call is one of the following, based on the return type:

```

typedef int64_t (* PluginCall)(void * data);
typedef double (* PluginCallD)(void * data);
typedef char16_t * (* PluginCallS)(void * data);

```

Synchronous calls may not delay the execution or wait for a longer period. The layout of the data buffer is specified with one character per parameter. The first character denotes the return type.

s	Zero terminated Unicode string
b	8 bit integer
w	16 bit integer
i	32 bit integer
l	64 bit integer
f	32 bit float
d	64 bit float
*	Array of one of the integral types. Provided as two elements, a 64 bit size and a 64 bit pointer.

Alignment of all elements is on their natural boundary. Return types may be either s, l or d.

Example:

sbb\*i

```

int16_t    w1;
int8_t     b1, b2;
int64_t    a1s;
int32_t *  a1p;

```

The memory of a returned string must be valid and thus cannot be placed on the stack of the called function. A guaranteed buffer of 500 bytes is provided at the end of the data object for this purpose. Longer strings must be allocated on the heap and maintained by the plugin.

## DSP Blocks

### Storage and Capture

#### TimeShift

Recording Start	recordstart(datetime) - readonly
Recording End	recordend(datetime) – readonly
Replay Start	replaystart(datetime)
Replay End	replayend(datetime)
Replay Position	replaypos(datetime) – readonly
Replay Speed	replayspeed(number, 1)
Recording	recording(bool, FALSE)
Replaying	replaying(bool, FALSE)
Passthrough	passthrough(bool, FALSE)
Replay No Flush	replaynoflush(bool, FALSE)
Replay Immediate	replayimmediate(bool, FALSE)
Stream Synch	streamsynch(bool, FALSE)
Buffer Size	bufferize(number, 1000000)
Buffer Used	bufferused(number) – readonly

#### FileWriter

Filename	filename(string)
Record	filerecord(bool, FALSE)
Auto Start	autostart(bool, FALSE)
Add Index	addfileindex(bool, FALSE)
Add Date	addfiledate(bool, FALSE)
Start Index	fileindex(number, 0)
Compression	compression(number, 1)
Previews	previewdist(number, 5.0)

### Temporal Processing and Resampling

#### TimeCompression

Time Compression	compression(number, 1)
Dynamic Full Range	dynamiccompression(bool, FALSE)
Compression Target	compressiontarget(number, 512)
Compression Mode	compressionmode(enum, Max, [Max;Average;Power;First;Min])



## TimeResample

Sample Rate	samplerate(number, 1000)
Interpolation	interpolationmode(enum, nearest, [nearest;average;linear;cubic;lanczos;gaussian;maximum;minimum])
Power Domain	powerdomain(bool, FALSE)
Time Offset	timeoffset(number, 0)

## TimeErosion

Mode	mode(enum, Erosion, [Erosion;Dilation;Opening;Closing])
Width	width(number, 0)

## TimeFIRFilter

Power	powerdomain(bool, FALSE)
Type	type(enum, average, [average;gaussian;sharpen;low pass;high pass])
Deviation	deviation(number, 0)

## TimeFFTFilter

Filter Mode	filtermode(enum, Low Pass, [Low Pass;High Pass;Band Pass;Low Reject;High Reject;Band Reject])
FFT Size	fftsize(number, 1024)
Resample Frequency	resamplefreq(number, 10000)
Lower Frequency	lowerfreq(number, 1000)
Upper Frequency	upperfreq(number, 1000)
Transition Frequency	transitionfreq(number, 100)
Pass Response	passresponse(number, 0)
Stop Response	stopresponse(number, -70)

## PulsedSegments

Samples	valuesteps(number, 512)
Persistence	persistence(number, 0.8)
Duration	buffertime(number, 0.016)
Jitter	jitter(number, 0)

## Arithmetic Processing

### LinearTransform

#### UnaryArithmetic

Offset	offset(bool, false)
Base	base(number, -90)
Mul	mul(number, 1)
Add	add(number, 0)
Min	min(number, -200)
Max	max(number, 2000)

#### CombiningArithmetic

Power	powerdomain(bool, FALSE)
Combine	combinemode(enum, max, [max;min;avg;add;sub;isub;diff;or;and;andnot;sqlch;select])
Combine Condition	combinecondmode(enum, first, [first;second;or;and;and not;equal;not equal])
Condition	condition[enum, C0, [C0;C1;C2;C3;!C0;!C1;!C2;!C3])
Lower Value	lowervalue(number, -120)
Upper Value	uppervalue(number, 20)

#### CombiningCondition

Combine Condition	combinecondmode(enum, first, [first;second;or;and;and not;equal;not equal])
-------------------	---

#### TriggerAndHold

Condition	tchannel[enum, C0, [C0;C1;C2;C3;C0,C1;C0,C2;C0,C3;C0,C1,C2;C0,C1,C2,C3])
Prefix	prefix(number, 100)
Postfix	postfix(number, 100)
Min Samples	minsamples(number, 1)
Max Skip	maxskip(number, 0)
Edge	edge(enum, low, [low;high;rising;falling;both])
Mode	mode(enum, free, [free;once;repeat;continuous;off])

#### SpectrumErosion

Mode	mode(enum, Erosion, [Erosion;Dilation;Opening;Closing])
Width	width(number, 0)

### FrequencyFIRFilter

Power	powerdomain(bool, FALSE)
Type	type(enum, average, [average;gaussian;sharpen;low pass;high pass])
Deviation	deviation(number, 0)

### SegmentSpectrum

Mode	mode(enum, percent, [percent;frequency;bin])
Start	startratio(number, 0)
Stop	stopratio(number, 1)

### SpectrumResample

Accumulation	accu(enum, power, [power;max;min])
Start Frequency	startfreq(number, 0)
End Frequency	endfreq(number, 1)
Bins	bins(number, 1024)

### SmoothNoiseFloor

Enable	smoothnoisefloor (bool. false)
Smooth Factor	smoothnoisefloorfactor (number, 0)

### PowerUnitConversion

### FrequencyUnitConversion

## Aggregation and Statistics

### MaxHold

Max Samples	maxsamples(number, 0)
-------------	-----------------------

Reset	maxreset(trigger)
-------	-------------------

#### AntennaMaxHold

Max Samples	maxsamples(number, 1)
-------------	-----------------------

#### MaxFall

Max Samples	maxsamples(number, 0)
-------------	-----------------------

#### MinHold

Max Samples	minholdsamples(number, 0)
Reset	minreset(trigger)

#### Average

Average Samples	avgsamples(number, 4096)
Average Power	avgpower(bool, FALSE)
Reset	avgreset(trigger)

#### Utilization

History Duration	historydur(number, 1)
Threshold Power	threspower(number, -60)

#### SpectrumOutline

Time Compression	maxsamples(number, 0.1)
Persistence	persistence(number, 0.95)
Sensitivity	sensitivity(number, 0.2)

#### PulsedSpectrumShape

Histogram Deviation	histdev(number, 16)
Histogram Min Density	histmindens(number, 0.1)

Frequency Span	freqspan(number, 2000000)
Persistence	persistence(number, 0.7)

### ChannelPower

Accumulate	accumulate(enum, power, [min;max;average;power;sum;center power])
Time Compression	timecompression(number, 1)
Antenna Segments	antennasegmentmode(enum, individual, [individual;merge;categories;overlap])

### DirectionPower

Accumulate	accumulate(enum, power, [min;max;average;power;sum;center power])
Time Compression	timecompression(number, 1)

### Histogram

Persistence Factor	persistence(number, 0.75)
Bins	valuesteps(number, 256)
Sampling	sampling(number, 0)
Display Mode	mode(enum, dots, [dots;contiguous;lines;fill;temporal])
Short Pulse Boost	shortboost(bool, false)

### SpectralHistogram

Persistence Factor	persistence(number, 0.9)
Bins	valuesteps(number, 256)
Sampling Rate	samplingrate(number, 8000)
Window Function	windowfunction(enum, Hamming, [Hamming;Hann;Uniform;Blackman;Blackman-Harris;Flat-Top;Lanczos])
Eliminate Harmonics	elimharmonics(bool, false)

### Detection

#### PulseDetector

Minimum Power Raise	minraise(number, 10)
Minimum Power	minpower(number, -60)
Minimum Total Power	mintotalpower(number, -40)

## IQ Processing

### IQNormalize

Auto Adjust Center	autocenter(bool, true)
Auto Adjust Power	autopower(bool, true)
Auto Adjust Circle	autocircle(bool, true)
Auto Adjust IQ Phase	autoiqphase(bool, true)
Target Power	power(number, -15)
Latency	latency(number, 0.5)
Phase Shift	phaseshift(number, 0)
IQ Phase Offset	iqphaseoffset(number, 0)

### IQToSpectrumConverter

FFT Size Mode	fftsizemode(enum, FFT, [FFT;Bins;Step Frequency;RBW;RBW FFT;Step Freq. FFT])
FFT Size	fftsize(number, 1024)
FFT Bin Size	fftbinsize(number, 1024)
FFT Step Frequency	fftstepfreq(number, 10000)
FFT RBW Frequency	fftrbwfreq(number, 10000)
FFT Overlap	fftoverlap(number, 0)
FFT Window	fftwindow(enum, Hamming, [Hamming;Hann;Uniform;Blackman;BlackmanHarris;Flat Top;Lanczos])
Power Range	powerrange(number, 90)
Drop Partial	droppartial(bool, true)
FFT Skip Windows	fftskip(number, 0)
Processing Latency	latency(number, 0.006)

### IQFrequencyFilter

Filter Type	filtertype(enum, Low Pass, [Low Pass;High Pass;Band Pass;Band Stop;Asymmetric])
Filter Window	filterwindow(enum, Hamming, [Rectangular;Bartlett;Hann;Hamming;Blackman])
Lower Frequency	lowfreq(number, 2000000)
Upper Frequency	highfreq(number, 5000000)
Filter Tabs	filtertabs(number, 7)
Center Frequency	centerfreq(number, 0)
Filter Gain	filtergain(number, 0)

## IQDemodulation

Center Frequency	centerfreq(number, 241000000)
Sample Rate	samplerate(number, 4000000)
Decim. Low Pass	slicelowpass(number, 1)
Span Frequency	spanfreq(number, 2000000)
Processing Latency	latency(number, 0.006)
Bounds Check	boundscheck(bool, true)

## IQToTimeSlice

Slice Duration	scliceduration(number, 0.001)
Slice Low Pass	slicelowpass(number, 1)
Trigger Level	triggerlevel(number, 0)
Trigger Mode	triggermode(enum, free, [free;once;repeat;off])
Trigger Source	triggersource(enum, i, [i;q;r])
Trigger Condition	triggercondition(enum, low, [low;high;rising;falling;both])
Trigger Duration	triggerduration(number, 0.000001)
Trigger Delay	triggerdelay(number, 0)
Trigger Blank	triggerblank(number, 0)

## IQHistogram

Persistence Factor	persistence(number, 0.75)
Mode	mode(enum, dots, [dots;contiguous;lines;fill;temporal])
Demodulation	demodmode(enum, none, [none;ofdm;ofdm_diff])
Phase Recovery	modphaserec(enum, none, [none;fixed phase;modulus power])
Modulation Frequency	modfreq(number, 0)
Modulation Phaseshift	modshift(number, 0)
Guard Type	guardtype(enum, blank, [blank;cyclic])
FFT Size	fftsize(number, 32)
Guard Samples	guardsamples(number, 16)
Band Limit Lower	bandlower(number, -4096)
Band Limit Upper	bandupper(number, 4095)
Band Equalization	bandequalization(number, 0)
Symbol Rotation	symbolrotation(number, 0)
Source X	sourcex(enum, i, [i;q;r;rq;phi;dphi;phase;band])
Source Y	sourcey(enum, q, [i;q;r;rq;phi;dphi;phase;band])
Value Bins	valuesteps(number, 256)

## IQHistogram3D

Persistence Factor	persistence(number, 0.75)
Mode	mode(enum, dots, [dots;contiguous;lines;fill;temporal])
Demodulation	demodmode(enum, none, [none;ofdm;ofdm_diff])
Phase Recovery	modphaserec(enum, none, [none;fixed phase;modulus power])
Modulation Frequency	modfreq(number, 0)
Modulation Phaseshift	modshift(number, 0)
Guard Type	guardtype(enum, blank, [blank;cyclic])
FFT Size	fftsize(number, 32)
Guard Samples	guardsamples(number, 16)
Band Limit Lower	bandlower(number, -4096)
Band Limit Upper	bandupper(number, 4095)
Band Equalization	bandequalization(number, 0)
Symbol Rotation	symbolrotation(number, 0)
Source X	sourcex(enum, i, [i;q;r;rq;phi;dphi;phase;band])
Source Y	sourcey(enum, q, [i;q;r;rq;phi;dphi;phase;band])
Source Z	sourcez(enum, phase, [i;q;r;rq;phi;dphi;phase;band])
Value Bins	valuesteps(number, 64)

#### IQCondition

Channel	tchannel(enum, [C0;C1;C2;C3])
Operation	toperation(enum, [Set;Set Not;Or;Or Not;And;And Not])
Hold	thold(number, 0)
Ignore	tignore(number, 0)
Minimum Power	minpower(number, 0.01)

#### IQSymbolDecoder

Symbol Coding	symcoding(enum, [none; auto; GFSK; BPSK; DBPSK; QPSK; QPSK_C; DQPSK; P4DQPSK; 8DPSK; QAM16; QAM64; QAM256; QAM1024; Bluetooth LE; Bluetooth; Bluetooth EDR 2M; Bluetooth EDR 3M; DECT; GSM; WiFi 802.11b; WiFi 802.11g 5MHz; WiFi 802.11g 10MHz; WiFi 802.11g 20MHz; Enhanced Shockburst:SiKRadioMAVLink])
Symbol Frequency	symfreq(number, 1000000)
Power Thresh Percentile	powerpercentile(number, 0.95)
Power Thresh Ratio	powerratio(number, 0.8)

#### IQSampleDelta

Sample Delta	sampledelta (enum, [bypass,sub,add,mul,rotate])
--------------	---



## Sweeping, Stitching and Segmentation

### SweepAggregate

Start Frequency	startfreq(number, 1000000)
Stop Frequency	stopfreq(number, 1000000)
Bin Mode	binmode(enum, Size, [Size,Count])
Bin Count	bincount(number, 4096)
Bin Size	stepfreq(number, 100000)
Log Frequency	logfreq(bool, false)
Sample Duration	sampledur(number, 0.03)
Hold Duration	holdduration(number, 0.02)
Aggregate Sweep	aggregate(bool, false)
Overlap Handling	overlap(enum, latest, [latest;max;min;average;floor;range])
Resample Mode	resample(enum, Power, [Power;Max;Min])

### SegmentPacker

### SpectrumStitcher

Adaptive Sync	adaptsync(bool, true)
Time Step	timestep(enum, any, [min;average;any;max])
Combiner	combindemode(enum, max, [max;min;avg;power;smooth])

### MergeCategories

### StreamSwitch

Outputs Detached	detached(bool, false)
Input	Input(number, 0, [0;1;2;3])
Outputs	output0(bool, false) output1(bool, false) output2(bool, false) output3(bool, false)

Select

Channel	tchannel(enum, [C0;C1;C2;C3])
---------	-------------------------------

SampleHold

Hold	hold(bool, false)
------	-------------------

Audio Processing

AudioMonitor

Volume	volume(number, 0.7)
Filter	filter(number, 7200)
Noise Reduction	squelsh(number, 0.1)
Mask Pilot	maskpilot(bool, false)

AudioMergeChannels

Drop Rejected	droprejected(bool, true)
---------------	--------------------------

AudioAMFMDecoder

Center Frequency	center(number, 9000000)
Frequency Span	span(number, 130000)
Modulation	modulation(enum, AM, [AM;FM])

AudioMultiAMFMDecode

AudioVoiceFilter

Energy	energybase(number, 1.4)
Persistence	persistence(number, 0.95)
Frequency	frequency(number, 0.375)
Threshold	threshold(number, 0.4)

## Video Processing

### ImageResize

Crop	crop(enum, none, [none;top left;top;top right;left:center;right;bottom left;bottom;bottom right])
Scale	scale(enum, none, [none;factor;width;height;free])
Crop Width	cropwidth(number, 512)
Crop Height	cropheight(number, 512)
Crop X Offset	cropxoffset(number, 0)
Crop Y Offset	copyoffset(number, 0)
Scale Width	scalewidth(number, 512)
Scale Height	scaleheight(number, 512)
Scale Factor	scalefactor(number, 1)

### ImageGreyscale

Source Channel	sourcechannel(enum, luminance, [luminance;avg;min;max;saturation;red;green;blue;alpha])
----------------	---

### ImageContrast

Absolute Value	absvalue(bool, FALSE)
Pre Scale	prescale(number, 1)
Offset	offset(number, 0)
Scale	scale(number, 1)
Threshold	threshold(bool, FALSE)
Lower Limit	lower(number, 0)
Upper Limit	upper(number, 1)

### ImageConvolution

Size	filtersize(enum, 3x3, [3x3;5x5])
Filter	filtermode(enum, identity, [identity;inverse;laplace 8+;laplace 8-;laplace 4+;laplace 4-;sharpen;box blur;gauss blur;sobel left;sobel right;sobel top;sobel bottom;mean distance;custom])

### ImageMorphologicalOperation

Operation	operation(enum, Erode, [Erode;Dilate;Open;Close;Outline;Median;Lower Bound;Upper Bound;Bound;Offset])
-----------	---

Size	size(number, 1)
------	-----------------

## VideoGlobalMotionDetection

### ImageFFTFilter

Filter Mode	filtermode(enum, Low Pass, [Low Pass;High Pass;Band Pass;Low Reject;High Reject;Band Reject])
Lower Frequency	lowerfreq(number, 1)
Upper Frequency	upperfreq(number, 1)
Transition Frequency	transitionfreq(number, 0.1)
Pass Response	passresponse(number, 0)
Stop Response	stopresponse(number, -70)

### ImageFrameDelta

Absolute Value	absolute(bool, FALSE)
Motion Compensation	motioncomp(bool, TRUE)
Expand Previous	expand(number, 0)

### ImageRegionLabeling

Min Area	minarea(number, 1)
Max Size	maxsize(number, 2048)

### ImageObjectTracking

Neural Network	nnetfile(string, )
----------------	--------------------

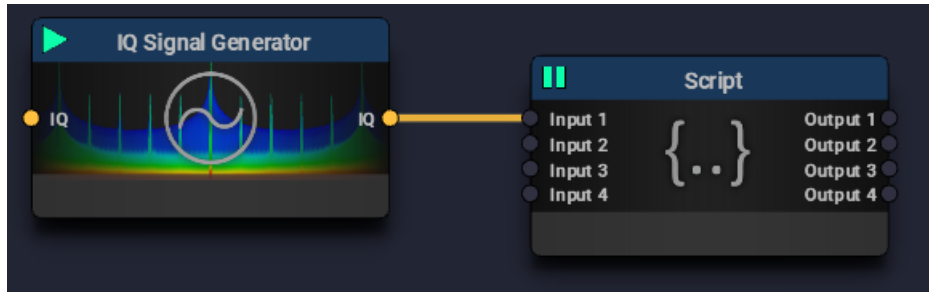
### ImageCombining

Combine	combinemode(enum, max, [max;min;avg;add;sub;isub;diff;or;and;andnot;sqlch;select;equal;not equal;greater equal;greater;less equal;less])
Condition	condition(enum, C0, [C0;C1;C2;C3;!C0;!C1;!C2;!C3])
Lower Limit	lower(number, 0)
Upper Limit	upper(number, 1)

## Samples

### Peak Detector

Continuously write the frequency and value of the peak signal in a spectrum to the console.



The data is provided by e.g. the signal generator block.

```
import { DSPStream } from "dspstream.js"

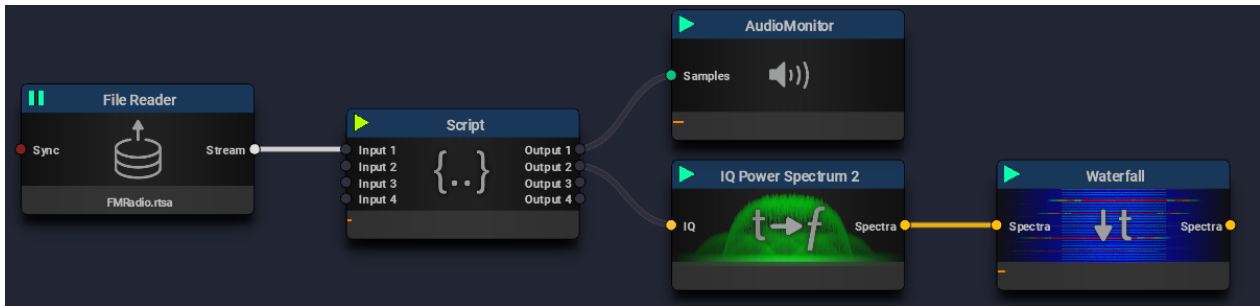
DSPStream.addBlocks({
  fft: {type: "IQToSpectrumConverter", config: {fftsize: 4096}}
}).then(() => {
  return DSPStream.connectBlocks([
    {source: "in0", drain: "fft"},
    {source: "fft", drain: "script", input: "in0"},
  ]);
}).then(() => {
  DSPStream.receivePackets(0, (flags, meta, samples) => {
    if (meta.samples) {
      let s = samples(0);
      let r = IQ.rankIndex(s, 1, true)[0];
      let f = meta.startFrequency + r * meta.stepFrequency;
      console.log(f, s[r]);
    }
  });
});
```

The script uses an IQ to spectrum converter block that uses an FFT with 4096 bins as a preprocessor. The output of the FFT block is forwarded to the script input connection 0.

The receive packet callback searches for the highest value in the first spectrum of each packet and dumps the frequency and signal strength. Using only the first spectrum reduces the rate of processing.

## FM Demodulator

The sample FM demodulator uses the phase change of the demodulated IQ samples as a measurement of the FM modulation.



The data is streamed from a file into the first input, demodulated and converted to audio. The result is displayed in a waterfall view and played as audio.

```
import { DSPStream } from "dspstream.js"

DSPStream.addBlocks({
  demodulator: {type: "IQDemodulation", config: {
    centerfreq: 100.395e6, samplerate: 192000, spanfreq: 48000}},
  sampleDelta: {type: "IQSampleDelta", config: {sampledelta: "rotate"}}
}).then(() => {
  return DSPStream.connectBlocks([
    {source: "in0",      drain: "demodulator"},
    {source: "demodulator", drain: "sampleDelta"},
    {source: "sampleDelta", drain: "script", input: "in0"},
    {source: "demodulator", drain: "out1"},
  ]);
}).then(() => {
  DSPStream.receivePackets(0, (flags, meta, samples) => {
    let s = samples();
    DSPStream.sendPacket(0, {
      payload: "audio",
      samples: IQ.mul(IQ.phi(s), 0.25),
      startTime: meta.startTime,
      endTime: meta.endTime
    });
  });
});
```

The script uses two preprocessing blocks, a demodulator to move the FM signal to the center and reduce the sample rate to audio rate and a sample delta block to convert the IQ samples to relative rotations.

Each sample is rotated by the complex conjugate of the previous sample and thus represents the phase change.

The blocks are connected in the second section and attached to the script input 0 and the block output 1 for inspection.

The receivePackets installs a callback, that is called with each incoming IQ packet. The IQ object uses the "phi" to convert all samples to their phase from  $-\pi$  to  $+\pi$  and multiplies them by 0.25 to get them into sample range. A new packet, now with audio data, is then sent to the block output 0 and played by the attached Audio Monitor block.